Web Services Proof-of-Concept

Project Report

Version 1.0

<06/19/2002>

Contact Information

Virginia Department of Alcoholic Beverage Control

Information Systems Division

2901 Hermitage Road

P. O. Box 27491

Richmond, Virginia 23261-7491

Telephone: 804-213-4709

Project Manager: Don Morgan

Revision History

Date	Version	Author	Description
	1.0		Initial Document Draft.

Table of Contents

1. Project Plan (Schedule)

Task – Assignment	**Anticipated Time (days)**	**Percentage Complete**	**Expected Completion Date**
Write Business Scenarios	1 day		6/25
Write Test Plan	1 day		6/28
Become familiar with XML file / schema	1 day	100%	
Become familiar with SOAP	1 –2 days		6/28
Become familiar with UDDI	1 day		6/28
Become familiar with JXM, JXRPC Protocols for java implementation of soap packages	3-4 days		6/28
Understand WSDL format	1 day		6/28
Learn to use web services with web logic	1 day		6/28
Team Meeting regarding business scenarios and Weblogic tools	0		TBD

Setup development environment	1 day		7/10
Use WSDL to write web services	1 day		7/19
Generate web services structure using WSDL	1 day		7/19
Implement methods / business logic	1 day		7/19
Deploy in weblogic application server and register services in UDDI	1 day		7/19
Team Meeting to review logic and to finalize implementation of services offered	0		TBD
			7/26
Write a client to look up UDDI web services	1 day		7/26
Use WSDL file to invoke web services of other parties	1 day		7/26
Implement methods / business logic	1 day		7/26
Testing	3 days		8/1
Team Meeting to review implementation of client side invocation of services and associated business logic	0		TBD
Prepare Presentation for Work Group Meeting on 8/5	1 day		8/1
Work Group Meeting and Presentations			8/5
Prepare COTS Report	TBD		8/26
Work group of draft reports	TBD		8/22
Submission of final report to COTS	TBD		8/26

2. List of Team Name, Members, Roles and Responsibilities

Name	Responsibility
David Hassen	Coordinate any resources required of the Virginia ABC to meet the desired objectives. Participate in the design and development of the web services components and integrate with a Virginia ABC test application.

Will Howery	Lead the technical and design efforts for implementing the web services using best practices and tools available from BEA. Participate in the prototype design and review development activities.
Dan Lender	Coordinate resources and supply BEA software necessary to implement the project.
Sue Shirbacheh	Participate in the design and development of the web services components and integrate with a Virginia ABC test application

3. Proof-of-Concept Design

See Appendix I

4. Business/Functional Requirements

4.1 Service Requirement One

The system will provide a subscription services that will take information pertaining to a type of address, a form of update mechanism (i.e. push or notify), and a callback URL (wsdl).

4.2 Service Requirement Two

The system will get subscribers, input address type, list of update type (push or notify) and a callback URL (wsdl).

4.3 General Requirements

- The system must invoke to subscribe noting the address type, update type, and callback URL.
- The system must get subscribers, their address type, a return list with the update mechanism (push or notify), and a call back URL.
- The system must invoke the callback URL (wsdl) for a notify.
- The system must invoke callback ULR (wsdl) for a push.
- The system must contain a method to get address type which will indicate if the type is an individual or business and a unique identifier.
- On notify, the system must indicate the type, individual business, and a unique identifier.
- On push, the system must indicate the type, individual or business, and the data.

4.4 Functional/Technical Assumptions

- The system will assume all vendors/contacts have Virginia Addresses for the purpose of this prototype.
- The system will assume that a company address change will be applied to all contacts for the given company.
- Design will begin under the assumption that the address schemas that will be published will be determined by the group.

5. Project Equipment Description

A BEA Weblogic application server will run on a windows 2000 PC which has a 1GHZ processor and approximately 30 gig hard drive. Oracle 8.17 personal database will be used. BEA Workshop will be used for the creation and deployment of the web services. The application will be a java based application.

6. Acquisition and Installation Activities

Additional hardware will not be required for this prototype. BEA has agreed to supply copies of the Weblogic application server and Workshop tool.

7. Server Requirements

BEA Weblogic application server will run on a windows 2000 PC which has a 1GHZ processor and approximately 30 gig hard drive. The purchase of additional resource sis not required. Server will need the follow items installed:

- BEA Weblogic
- BEA Workshop
- JXM, JXRPC
- Oracle 8.17

8. Networking Requirements

The server and entire application environment will run outside of the VABC firewall. This will require the system administrator to setup a unique IP address for this machine. No additional networking requirements are anticipated.

9. Training Requirements

Training Requirements

- Self Study – XML
- Self Study – SOAP
- Self Study – UDDI
- Self Study – JXM, JXRPC
- Self Study – WSDL
- BEA assisted – Weblogic / Web Services Integration
- BEA assisted – Workshop
- BEA assisted – technical guidance

10. Testing Plan

Attach a list showing all functions/processes to be tested. This should include descriptions of all testing scenarios and results.

11. Development/Technical

High-level description of specific development/technical issues and/or difficulties encountered during coding.

12. Protocol/Specification Issues

Issues and/or difficulties specific to designing and coding for XML, SOAP, WSDL, and UDDI.

13. Interoperability Issues

Discuss issues and/or difficulties that involve service-to-service interoperability between the various team applications.

14. “Best Practice” Comments

Technical or non-technical.

15. Other Concerns/Issues

- UDDI must be implemented early in the process.
- Subscription services must be implemented early in the process.
- Address formats need to be identified.
- XML structures (list information) must be identified.
- Test data will need to be coordinated between teams.
- Unique identifier should be defined for companies.
- Unique identifier should be defined for individuals.
- SSN identifier may possibly be needed for a sole proprietor.

16. Other Comments

Comments on how the team project is going, successes, etc.

17. Cost/Time Estimate

Track the time the full team spends for 1) Meetings, 2) Development, 3) Training, and 4) Testing.

Appendix I — Proof of Concept Design

This Appendix outlines the basic interfaces and data representations needed to implement the address change and update Web Services project.

Terms

- **Address type —** refers to the types of address that is being updated, pushed, or registered for.

- **Address Identifier (ID) —** The address identifier is a unique identifier for a given address. This ID allows correlation of an updated address from agency to agency and system to system. Without a defined address ID there would be duplication of addresses and names.

- **Entity type —** refers to the individual or a business entity. The reason to delineate the type of entity is to determine the address ID. The presumption is that if the address entity is an individual then the social security number of that individual can be used as the identifier of the address. If the address entity is a business then the address identifier could be the federal tax ID. The only businesses that do not have federal tax ids are sole proprietors, and then the social security number of the owner could be used.

- **Notification or Push —** Agencies can select if they want addresses sent directly to them (push), or if they want to be notified of the address changed and then have the ability to go back and access the address at the time of their choosing.

Class diagram of interfaces and data definitions

The following class diagram details the interfaces for the subscription service and the data elements needed for an address representation and subscriber list.

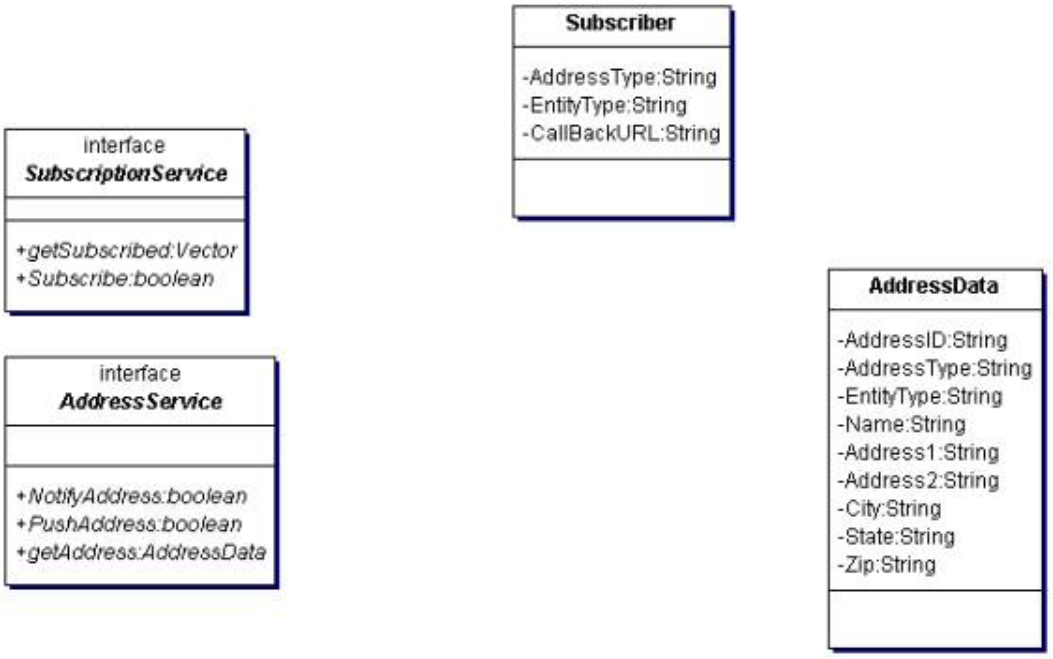

Figure 1. Address WebServices class diagram

The subscriber class represents the information that will be stored by the subscription service, and subsequently returned in a list of subscribers when an agency invokes the getSubscriber service on the subscription service. The address Data class represents the data and id data for an address.

```
<subscriberlist>

<subscriber addresstype='billing'
   entitytype='business'>

http://whowery:7001/webservice/NotifyAddress.wsdl

</subscriber>
```

```
<subscriber addresstype='billing'
   entitytype='business'>

http://vabc:7001/webservice/PushAddress.wsdl

</subscriber>

</subscriberlist>
```

Figure 2. Suggested XML representation of a subscriber list

Address XML schema representation. The following structure represents a sample generic address xml document.

<address>

<addresstype>

billing

</addresstype>

<addresseid>

FEDIDXXXX

</addressed>

<timestamp>

xsd:dateTime

</timestamp>

<source>

VABC

</source>

<address1>

Appt 123

</address1>

<address2>

12529 Swallow Creek rd

</address2>

<city>

Richmond

```
  </city >
  <state>
        VA
  </state >
  <zip>
        23233
  </zip>
</address>
```

addresstype (mandatory): The type of address e.g: home, business, billing, garage, mailing

addressid (mandatory): identifier for an address typically the users social security number or a businesses tax id number 9 numeric digits in length.

timestamp xsd:dateTime: time that the address was last updated. Used in the log and query results

source: source URL of address change. Used in the log and query results.

address1 (mandatory): first line of an address. alphanumeric

address2: second line of an address. alphanumeric

city (mandatory): city name. Alphanumeric

state (mandatory): 2 character state abbreviation

zip: 5 digit zip code, optionally could contain 9 digit zip code with a hyphen. NNNNN-NNNN

When an address xml document is received by an agency that document will be handled in the most appropriate mechanism for that agency. The agency will also keep an operation log of the document. This log will fulfill two purposes. One it will provide the bases for sending back address changes to a queryAddress request and two, it will provide a logging instance to test and validate that the service received all addresses sent to it. After the address change WebServices have been deployed a test can be run with all changes going to a log. This log can then be analyzed to validate that each agency had in fact propagated address changes and received any address changes propagated.

```
<addresslog>
<address>
  <addresstype>
        billing
  </addresstype>
  <addresseid>
        FEDIDXXXX
  </addressed>
```

<timestamp>

xsd:dateTime

</timestamp>

<source>

VABC

</source>

<address1>

Appt 123

</address1>

<address2>

12529 Swallow Creek rd

</address2>

<city>

Richmond

</city >

<state>

VA

</state >

<zip>

23233

</zip>

</address>

<address >

.

.

.

</address>

</addresslog>

The addresslog schema example will contain multiple address entries. Reusing the address structure will reduce the effort in managing the log entries. The only difference in the address log entry from the address schema will be the mandatory addition of a source and a time stamp. The source will identify the url which sent this message to the receiving agency. The time stamp will be in the Year/Month/Day Hour: Minute:Second format as yyyy/mm/dd hh:mm:ss. For purposes of the testing we can use Virginia local time. If the system were to be expanded we would want to record GMT or include a time zone identifier.

WSDL document:

The following WSDL implementation will represent the interfaces discussed in the workshop meeting. As follows:

```
public interface AddressChangeWebService {

public AddressData getAddress(String addressID, String addressType)  throws
        InvalidAddressID;

  public boolean changeAddress(AddressData aAddressData, String OriginatorURL);

  public AddressLog queryAddress(Date startDate, Date endDate,  String addressType, String
         OriginatorURL);
```

The WSDL document defines the AddressData XML structure. The AddressLog coming from the queryAddress still needs some work to not be more portabl e.

```
  <?xml version="1.0" encoding="utf-8" ?>

<definitions xmlns:s="http://www.w3.org/2001/XMLSchema" xmlns:http="http://schemas.xmlsoap.org/wsdl/http/ " xmlns:soap="http://
schemas.xmlsoap.org/wsdl/soap/" xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/" xmlns:tns="http://whowery:7001"
targetNamespace="http://whowery:7001" xmlns="http://schemas.xmlsoap.org/wsdl/">

<types>

<xsd:schema attributeFormDefault="qualified" elementFormDefault="qualified" targetNamespace="java:language_builtins.util" xmlns:
xsd="http://www.w3.or g/2001/XMLSchema" xmlns:stns="java:language_builtins.util">

<xsd:complexType name="Vector">

<xsd:complexContent>

<xsd:restriction base="soapenc:Array" xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/">

  <xsd:attribute ref="soapenc:arrayType" wsdl:arrayType="xsd:anyType[]" xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/" />

  </xsd:restriction>

  </xsd:complexContent>

  </xsd:complexType>

  </xsd:schema>

<xsd:schema attributeFormDefault="qualified" elementFormDefault="qualified" targetNamespace="java:com.bea.sesouth.webservices"
xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:stns ="java:com.bea.sesouth.webservices">

  <xsd:import namespace="java:language_builtins.util" />

<xsd:complexType name="AddressLog">
```

```
<xsd:sequence>
  <xsd:element name="addressVector" maxOccurs="1" type="tp:Vector" minOccurs="1" nillable="true" xmlns:tp="java:
language_builtins.util" />
  <xsd:element name="originator" maxOccurs="1" type="xsd:string" minOccurs="1" nillable="true" />
  </xsd:sequence>
  </xsd:complexType>
<xsd:complexType name="AddressData">
<xsd:sequence>
  <xsd:element name="originator" maxOccurs="1" type="xsd:string" minOccurs="1" nillable="true" />
  <xsd:element name="address2" maxOccurs="1" type="xsd:string" minOccurs="1" nillable="true" />
  <xsd:element name="state" maxOccurs="1" type="xsd:string" minOccurs="1" nillable="true" />
  <xsd:element name="name" maxOccurs="1" type="xsd:string" minOccurs="1" nillable="true" />
  <xsd:element name="zip" maxOccurs="1" type="xsd:string" minOccurs="1" nillable="true" />
  <xsd:element name="address1" maxOccurs="1" type="xsd:string" minOccurs="1" nillable="true" />
  <xsd:element name="addressID" maxOccurs="1" type="xsd:string" minOccurs="1" nillable="true" />
  <xsd:element name="city" maxOccurs="1" type="xsd:string" minOccurs="1" nillable="true" />
  <xsd:element name="updateDate" maxOccurs="1" type="xsd:dateTime" minOccurs="1" nillable="true" />
  <xsd:element name="addressType" maxOccurs="1" type="xsd:string" minOccurs="1" nillable="true" />
  </xsd:sequence>
  </xsd:complexType>
  </xsd:schema>
  </types>
<message name="changeAddress">
  <part name="addressData" xmlns:partns="java:com.bea.sesouth.webservices" type="partns:AddressData" />
  <part name="string" xmlns:partns="http://www.w3.org/2001/XMLSchema" type="partns:string" />
  </message>
<message name="changeAddressResponse">
  <part name="result" xmlns:partns="http://www.w3.org/2001/XMLSchema" type="partns:boolean" />
  </message>
```

```
<message name="getAddress">
  <part name="string" xmlns:partns="http://www.w3.org/2001/XMLSchema" type="partns:string" />
  <part name="string0" xmlns:partns="http://www.w3.org/2001/XMLSchema" type="partns:string" />
  </message>
<message name="getAddressResponse">
  <part name="result" xmlns:partns="java:com.bea.sesouth.webservices" type="partns:AddressData" />
  </message>
<message name="queryAddress">
  <part name="date" xmlns:partns="http://www.w3.org/2001/XMLSchema" type="partns:dateTime" />
  <part name="date0" xmlns:partns="http://www.w3.org/2001/XMLSchema" type="partns:dateTime " />
  <part name="string" xmlns:partns="http://www.w3.org/2001/XMLSchema" type="partns:string" />
  <part name="string0" xmlns:partns="http://www.w3.org/2001/XMLSchema" type="partns:string" />
  </message>
<message name="queryAddressResponse">
  <part name="result" xmlns:partns="java:com.bea.sesouth.webservices" type="partns:AddressLog" />
  </message>
<portType name="AddressChangeBeanPortType">
<operation name="changeAddress">
  <input message="tns:changeAddress" />
  <output message="tns:changeAddressResponse" />
  </operation>
<operation name="getAddress">
  <input message="tns:getAddress" />
  <output message="tns:getAddressResponse" />
  </operation>
<operation name="queryAddress">
  <input message="tns:queryAddress" />
  <output message="tns:queryAddressResponse" />
```

```
  </operation>
  </portType>
<binding name="AddressChangeBeanSoapBinding" type="tns:AddressChangeBeanPortType">
  <soap:binding style="rpc" transport="http://schemas.xmlsoap.org/soap/http" />
<operation name="queryAddress">
  <soap:operation soapAction="" />
<input>
  <soap:body use="encoded" namespace="http://whowery:7001" encodingStyle="http://schemas.xmlsoap.org/soap/encoding/" />
  </input>
<output>
  <soap:body use="encoded" namespace="http://whowery:7001" encodingStyle="http://schemas.xmlsoap.org/soap/encoding/" />
  </output>
  </operation>
<operation name="changeAddress">
  <soap:operation soapAction="" />
<input>
  <soap:body use="encoded" namespace="http://whowery:7001" encodingStyle="http://schemas.xmlsoap.org/soap/encoding/" />
  </input>
<output>
  <soap:body use="encoded" namespace="http://whowery:7001" encodingStyle="http://schemas.xmlsoap.org/soap/encoding/" />
  </output>
  </operation>
<operation name="getAddress">
  <soap:operation soapAction="" />
<input>
  <soap:body use="encoded" namespace="http://whowery:7001" encodingStyle="http://schemas.xmlsoap.org/soap/encoding/" />
  </input>
<output>
  <soap:body use="encoded" namespace="http://whowery:7001" encodingStyle="http://schemas.xmlsoap.org/soap/encoding/" />
```

```
  </output>
  </operation>
  </binding>
<service name="AddressChangeBean">
  <documentation>todo: add your documentation here</documentation>
<port name="AddressChangeBeanPort" binding="tns:AddressChangeBeanSoapBinding">
  <soap:address location="http://localhost:7001/state.va.ws.addresschange/AddressChangeBean?WSDL" />
  </port>
  </service>
  </definitions>
```

Figure 3. Suggested XML representation of a subscriber list

UDDI Registration

Agencies and the subscription service will register in a UDDI registry. This registry will contain information about the services exposed and a Web Services Description Language (WSDL) document that contains the necessary details for other agencies to access and invoke the various address web services.

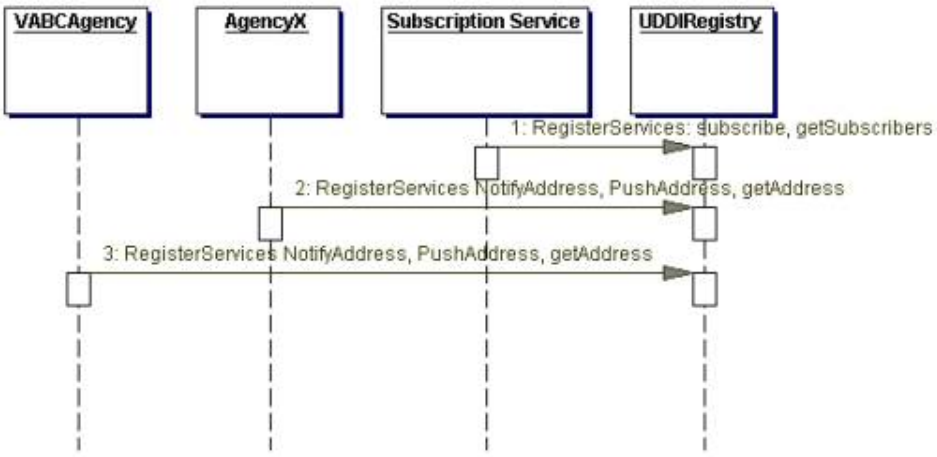

Figure 4. Agencies registering Web Services with the UDDI

Subscription Services

An agency subscribing for address changes is the first step in Address Web Services integration: An agency subscribes with the address subscription service. The agency supplies the type of address it wants to be updated with, the manner of being updated (e.g. notification or push), the type of entity (e.g. individual or company) and a call back URL for the WSDL that will

provide the update service. The subscription service will keep this information and return the subscriber list to any other service that request the supplied address type and entity type.

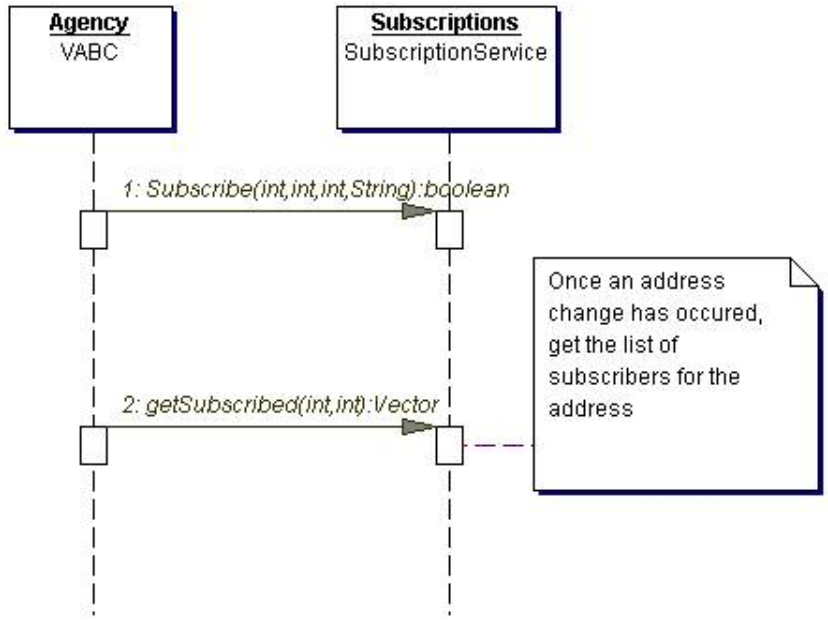

Figure 5. Subscribing to the subscription service, and getting the list of subscribers

Signature for subscribing:

```
boolean Subscribe(String addressType, String entityType, String
updateType, String callBackURL);
```

For the second subscription service, An agency has had an address change and now wants to ask the subscription service for the list of agencies that have subscribed to updates for this type of address change. The agency will send the address type, and entity type to the getSubscription service and the subscription service will return a list of subscribers with the mode of update and callback URLs for the services that will be used to update the subscribing agencies.

Signature for getSubscribed:

```
Vector getSubscribed(String addressType, String entityType);
```

Processing a Change In Address

The following scenario details the actions that will occur when an address is changed.

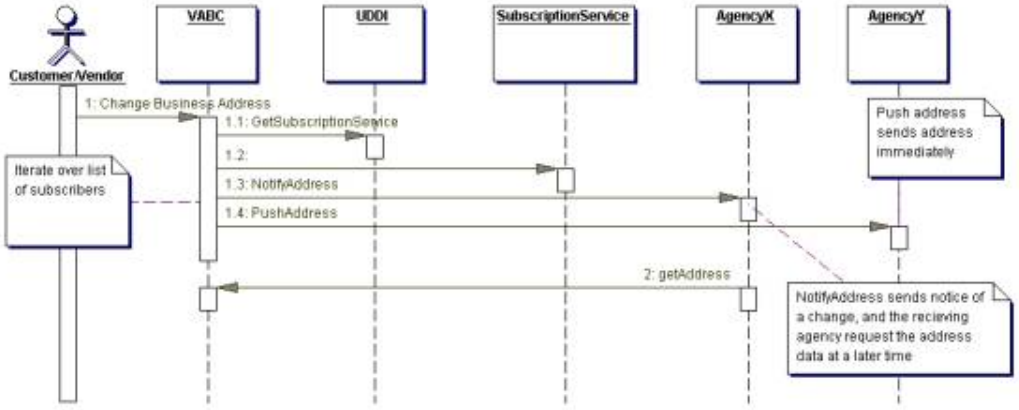

Figure 6. Address change scenario with NotifyAddress, PushAddress and getAddress.

When a customer or Vendor changes an address with a host agency that agency will start the process of sending updates or notifications of address changes to other agencies that have subscribed for that type of address change. The first step for the host agency will be to check in the UDDI and validate the subscriber service. Then the host agency will invoke the getSubscriber service on the subscription WebService. The getSubscriber service will pass in the address type, and entity type information. From this information the subscription service will search for subscribers who have subscribed for that address type with that entity type. The subscription service will create a list of subscribers (note Fig 1; for the Subscriber object content) this list will be returned to the host service. When returned the internal object list will be serialized to an XML document for the host service to consume and process.

Signature of the getSubscriber service:

```
Vector getSubscribed(String addressType, String entityType);
```

Once the host service has the list of subscribers the host service will iterate through the list and either notify or push the address change to the subscribed agencies. When a Notify address is invoked the host service will send the address type entity type, address id and a call back URL for the subscribing agency to use when later requesting the address data from the host agency.

Signature of NotifyAddress:

```
boolean NotifyAddress(String addressType, String entityType, String addressID, String
CallbackURL);
```

When a push address is requested from the subscribed agency, the host agency will immediately send the address data to the subscribing agency.

Signature of the PushAddress:

```
boolean PushAddress(String addressType, String entityType, String addressID, AddressData
addressData);
```

At some later time the agency that received the NotifyAddress will need to request that address information from the host agency. The purpose of the NotifyAddress services is to allow agencies to load balance their communications and selectively choose when they well receiving address data, and updating their systems. The subscribing agency will store the notification information and at a later time invoke a getAddress on the host agency to retrieve the actual address data.

Signature of the getAddress service:

```
AddressData getAddress(String addressType, String entityType, String addressID);
```

This document represents the result of analyzing the data and signatures needed to implement the address change functionality across agencies. The document does not prescribe how to implement the security features needed for these services. The presumption is that the Web Services clients will authenticate them selves with the agency services servers and invoke the said services via HTTPS with SSL security.

Supplemental Information

The following diagrams show existing database and class relationships that already existing within the Management of Inventory and Product Sales (MIPS) System at the Virginia ABC. The VABC-BEA team hopes to use these existing structures for purposes of the prototype.

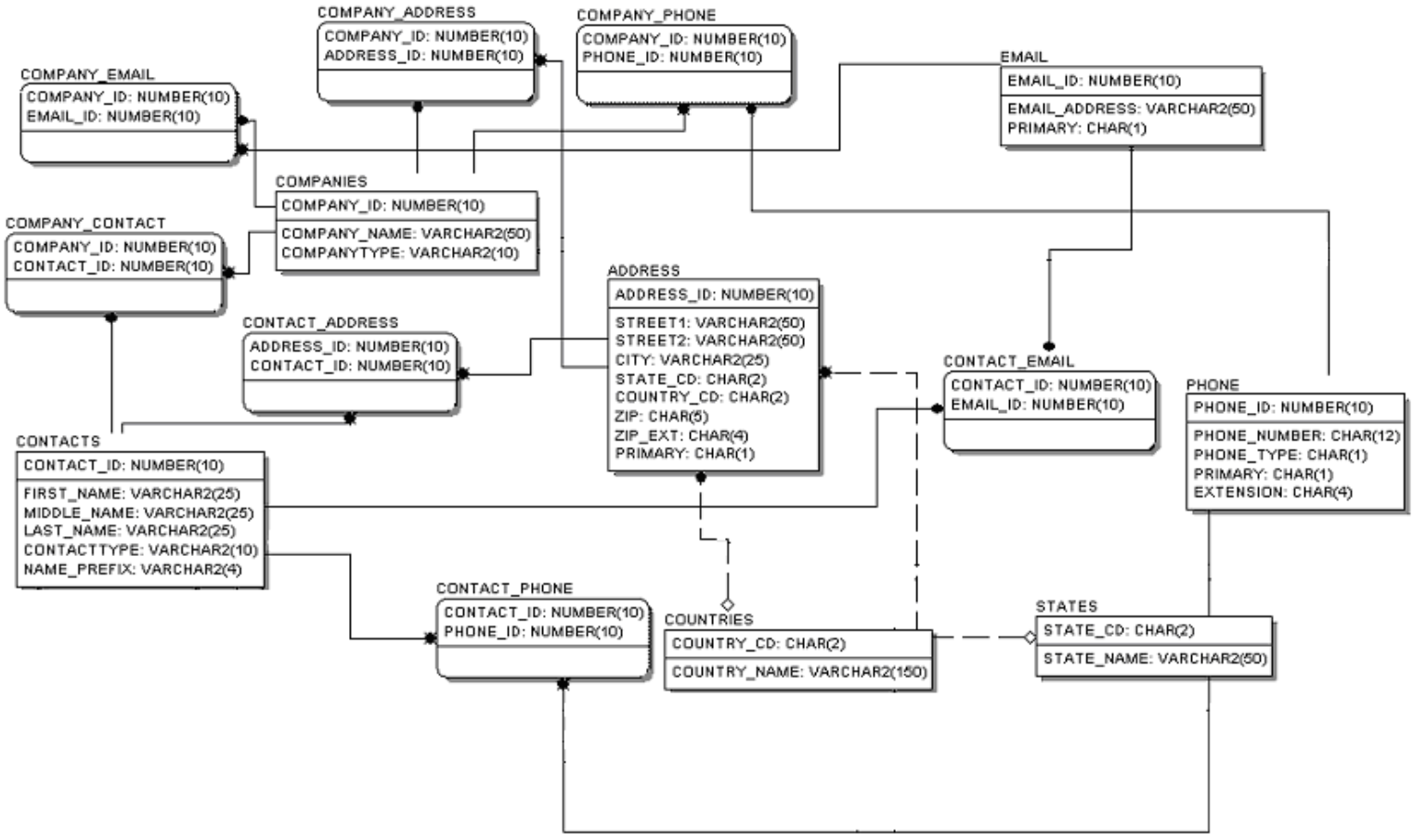

Figure 7. Contact Database Schema

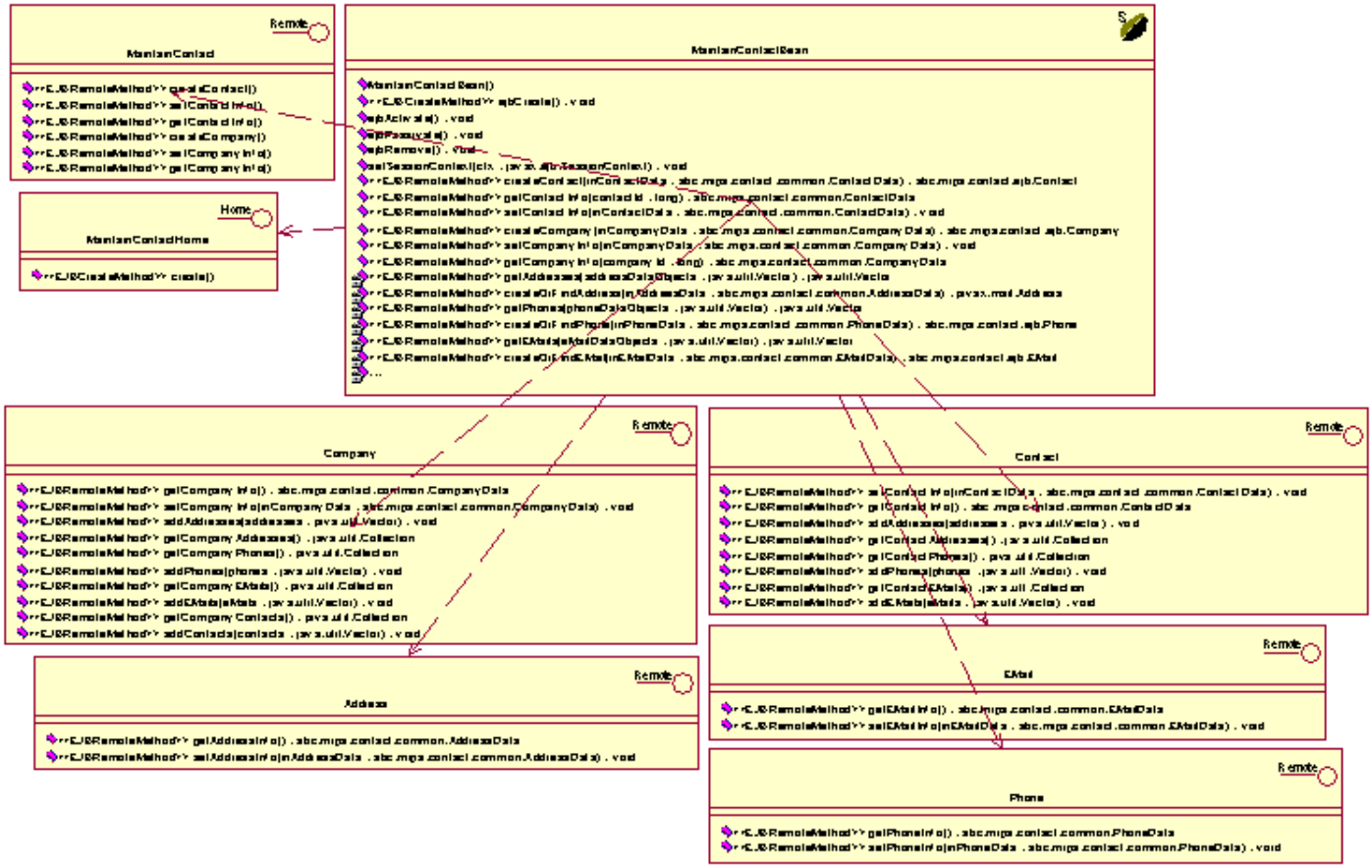

Figure 8. Contact Class Diagram

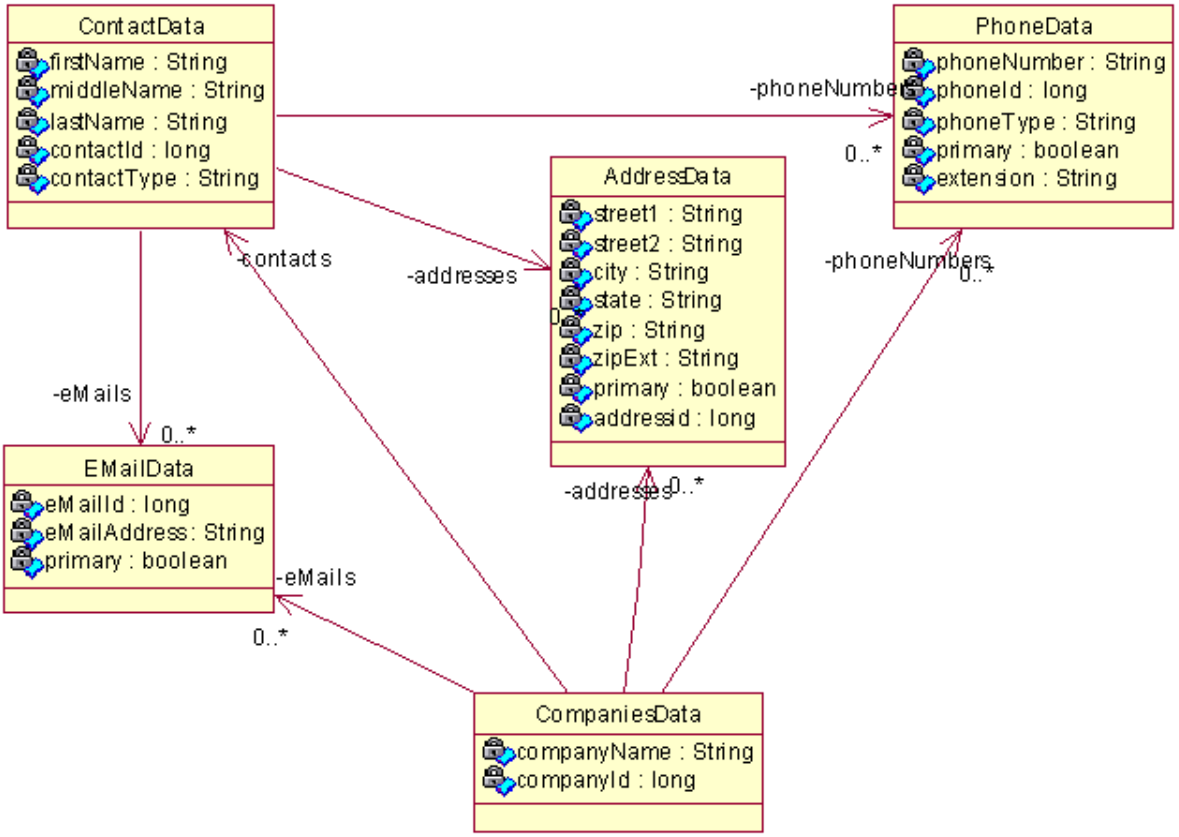

Figure 9. Contact Value Objects